International Journal of Research
*eISSN: 2348-6848* & *pISSN:* 2348-795X *Vol-5 Special Issue-13*
International Conference on Innovation and Research in Engineering, Science & Technology
Held on 23rd & 24th February 2018, Organized by Tulsiramji Gaikwad Patil College of Engineering & Technology, Nagpur, 441108, Maharastra, India.

# Optimizing File Transfer Mechanism in Distributed computing by weighted object Subsystems

**Madhav Ganpat Raut[1]& Pradeep B.Dahikar[2],**
[1] Department of Electronics Hislop College, Nagpur
[2] Head of, Department of Electronics. Kamla Nehru Mahavidalaya,Nagpur
mahavraut63@gmail.com,pbdahikarans@reddifmail.com

***Abstract:***
*In this paper, we introduce a high-performance file transfer mechanism that is optimized for the high-bandwidth, high-delay networks currently being developed and implemented for computational Distributed. We discuss the file transfer mechanism, the underlying communication protocol upon which it is based, and our plans for more generalized file transfer services. Also, we provide preliminary performance results demonstrating that our approach is able to achieve excellent performance in a computational Distributed environment.*
***Keywords:-*** *Globs, FTP, Design of LOBS, It is layered on top of a lower-level data movement protocol (FOBS)*

## 1. Introduction

A significant amount of current research is aimed at the development, implementation, and testing of the cutting-edge networking infrastructure of the future (e.g. the Internet2 initiative [18] and the Illinois WIRE initiative [19]). An integral aspect of such research efforts is the development and testing of the high-performance distributed applications that, due to the limited bandwidth and best-effort nature of the Internet1 environment, were heretofore infeasible. Examples of such applications include distributed collaboration, remote visualization, distributed supercomputing, Internet telephony, and advanced multimedia environments. Such high-bandwidth, high-delay networks, capable of delivering data at speeds up to 40 Gigabits per second, are quickly becoming a significant component of the national computational infrastructure.

At the heart of any such computational Distributed environment (i.e. geographically distributed computational resources connected by very high-speed networks) is the ability to transfer very large amounts of data in a very efficient manner. All of the developing and envisioned advanced distributed applications are predicated upon this fundamental ability. However, it has been well established that in practice the actual bandwidth achieved by distributed applications executing in a Distributed environment (e.g. the Internet2 infrastructure) represents only a very small fraction of the available bandwidth [4,5,6]. The problem is that TCP, the data transfer mechanism of choice for Distributed-based computations, has been shown to perform very poorly in a high-bandwidth, high-delay network environment [4,5,6,13,14,15]. Given the performance problems inherent within the TCP protocol, there is a significant amount of current research aimed at developing more effective techniques for delivering data across high-performance computational Distributed.

In this paper, we describe LOBS (Lightweight Object-Based file transfer System), a mechanism to transfer large files between computational resources in a computational Distributed. This mechanism is lightweight in the sense that it does not attempt to support all of the functionality of, for example, the DistributedFTP [1] protocol developed for the Globus meta-computing environment [16]. It supports the primary functionality required for computational Distributed (e.g. fast, robust file transfers), but is not built upon and does not require the complete infrastructure of a meta-computing environment. It is object-based in the sense that *none* of the data being transferred is considered correct until *all* of the data has been delivered. Thus the order in which the data is delivered does not matter (in contrast to stream-based mechanisms such as TCP where the data must be delivered in exactly the same order in which it was sent), allowing the data transfer mechanism to be optimized for performance.

The rest of this paper is organized as follows. In Section 2, we discuss the work most closely related to our own. In Section 3, we provide an overview of LOBS and the data transfer mechanism upon which it is based. In Section 4, we provide preliminary performance results taken between two sites connected by the Abilene backbone network. In Section 5, we discuss the functionality of the high-level graphical user interface being developed as part of this project, and we provide our conclusions in Section 6.

## 1 Related Work

There has been a significant amount of work related to achieving the full bandwidth available in a high-performance high-delay network environment. There are two approaches currently being pursued: One approach is to modify the TCP protocol itself to overcome its performance problems within this network environment. The other approach is to develop user-level mechanisms that can circumvent the performance problems inherent within the protocol. We briefly discuss each of these approaches in turn.

As discussed in [13,14,15], the size of the TCP window is the single most important factor in achieving good performance over high-bandwidth, high-delay networks. To keep such "fat" pipes full, the TCP window size should be at least as large as the product of the bandwidth and the round-trip delay. This has led to research in automatically tuning the size of the TCP socket buffers at runtime [15]. Also, it has led to the development of commercial TCP implementations that allow the system administrator to significantly increase the size of the TCP window to achieve better performance [13]. Another area of active research is the use of a Selective Acknowledgement mechanism [9,17] rather than the standard cumulative acknowledgement scheme. In this approach, the receiving TCP sends to the sending TCP a Selective Acknowledgement (SACK) packet that specifies exactly those packets that have been received, allowing the sender to retransmit only those segments that are missing. Additionally, "fast retransmit" and "fast recovery" algorithms have been developed that allow a TCP sender to retransmit a packet before the retransmission timer expires, and allows the TCP sender to increase the size of its congestion control window, when three duplicate acknowledgement packets are received (without intervening acknowledgements) [9,17]. An excellent source of information, detailing which commercial and experimental versions of TCP support which particular TCP extensions, may be found in [13].

The second general approach is to attempt to overcome the performance problems of TCP at the user level. The DistributedFTP protocol [1] developed by Globus [16] allocates multiple TCP streams per file transfer (and per host if it is being transferred to/from a parallel file system). Allocating multiple TCP streams per data flow provides significant performance benefits for two reasons: First, it creates an aggregate TCP buffer size that is closer to the ideal size of the product of the bandwidth and delay of the network. Secondly, it essentially circumvents the congestion control mechanisms implemented in the TCP protocol. That is, as the number of TCP streams increases the probability that *all* such streams are blocked due to congestion control mechanisms decreases. Thus it is likely that at any given time there is at least one TCP stream that is ready to fire.

Our approach differs in that we use a single UDP stream as the basic underlying transport mechanism. We chose this approach for three reasons:

First, the buffers are allocated at the user level rather than at the kernel level. Secondly, it avoids multiplexing multiple TCP streams at the kernel level. Thirdly, it provides opportunities for user-level enhancements that are not possible with kernel-level algorithms such as TCP (without kernel-level permissions). However, it is also very important to note that the Globus DistributedFTP protocol has functionality greater than that supported by LOBS. In particular, the DistributedFTP protocol also supports the creation and manipulation of replicas of large datasets and mechanisms for maintaining a catalog of such dataset replicas. These are of course very important issues, but are viewed as outside of the domain of this work that is focused only on the fast and reliable transfer of very large files.

Other data transfer mechanisms similarly employ a single user-level UDP stream. The two most closely related to our own are RUDP [7] and SABUL [12]. The primary difference between our approach and SABLE is how packet loss is interpreted and how such loss impacts the behavior of the protocol. In particular, SABLE assumes packet loss indicates congestion, and it reduces the sending rate based on this perceived congestion. Our approach assumes some packet loss is inevitable in wide area transfers, and that such loss does not necessarily indicate congestion and does not necessarily require a reduction in the sending rate. The primary difference between LOBS and RUDP has to do with the types of networks for which the protocol is designed. In particular, RUDP is designed for very reliable networks with QoS guarantees [7]. Our approach has been shown to achieve excellent performance on non QoS-enabled networks with minimal additional network load (that is, minimal wasted network resources) [2,3]. This discussion will be expanded in the final paper.

## 3 LOBS

The file transfer mechanism is built directly on top of FOBS: A Fast Object-Based data transfer System. FOBS is described in detail elsewhere [2,3], and here we only touch upon its basic features. As noted above, FOBS is an *object-based* data transfer system. By object-based, we mean that none of the data transferred can be assumed correct until all of the data has been received. This is in contrast to stream-based protocols such as TCP where each byte of data must be delivered to the application in exactly the same order in which it was sent. Streams are kernel-level constructs and are implemented at the kernel level. Objects on the other hand are user-level constructs and are implemented at the user level. This allows knowledge of the characteristics of the data transfer itself to be leveraged to significantly enhance performance.

The fundamental characteristic of an object-based data transfer that is leveraged by FOBS is the assumption that the user-level data buffer spans the entire object to be transferred. For the moment, assume that this is correct and consider how it can be leveraged to enhance performance. In particular, this characteristic allows FOBS to push to the limit the basic concept of the "Large Window" extensions developed for TCP: that is, the window size is essentially infinite since it spans the entire data buffer (albeit at the user level). It also pushes to the limit the idea of selective acknowledgements. Given a pre-allocated receive buffer and constant packet sizes, each data packet in the entire buffer can be numbered. The data receiver can then maintain a very simple data structure with one byte (or even one bit) allocated per data packet, where this data structure tracks the received/not received status of *every* packet to be received. This information can then be sent to the data sending process at a user-defined acknowledgement frequency. Thus the selective acknowledgement window is also in a sense infinite. That is, the data sender is provided with enough information to determine *exactly* those packets, across the *entire* data transfer, that have not yet been received (or at least not received at the time the acknowledgement packet was created and sent). As discussed in [2,3], these features of the data transfer mechanism result in excellent performance in a high-bandwidth, high-delay network environment, obtaining most of the available bandwidth with minimal wasted network resources.

However, the assumption that the entire object can be maintained in a user-level buffer clearly does not apply to the transfer of terabyte or larger files. Thus the file transfer mechanism is layered on top of FOBS, supplying it with (user-defined) "chunks" of data that *can* be buffered at the user level. The underlying data transfer mechanism then works under the assumption that each individual buffer is complete object, and applies the basic object-based data transfer mechanism for each such chunk.

Figure 1 provides an overview of the design of LOBS and how it interacts with the underlying data transfer mechanism. First, consider the data sender. LOBS allocates some number of threads, each of which controls its own data buffer, to read the file from the disk into the user data area. Once a buffer is filled, it is passed to FOBS to transfer over the network. The basic idea is that while this network data transfer is progressing, the other threads are concurrently reading data off of the disk into their local buffers. The goal is to overlap the network I/O operation with the disk I/O operation to the largest extent possible.

This basic algorithm is reversed in the case of the data receiver. In this case, FOBS reads the data off of the network into a local buffer. Once a buffer is filled completely (i.e. the entire object, from the point of view of FOBS, has been successfully transferred), a writer thread wakes up and transfers the data from the buffer to the file. As this write to disk is progressing, the data transfer mechanism begins a new object transfer into one of the other local buffers. The "optimal" number of threads and buffers to be employed is an open issue currently under consideration. It is also important to note that since the underlying data transfer mechanism is robust, the overall file transfer mechanism is also robust. If the application crashes, the maximum amount of lost data is the sum of the data written into the buffers not yet flushed to disk, and the data in the network at the time of the crash. Very little additional information has to be maintained to determine the correct state of the file transfer once the system is restarted.

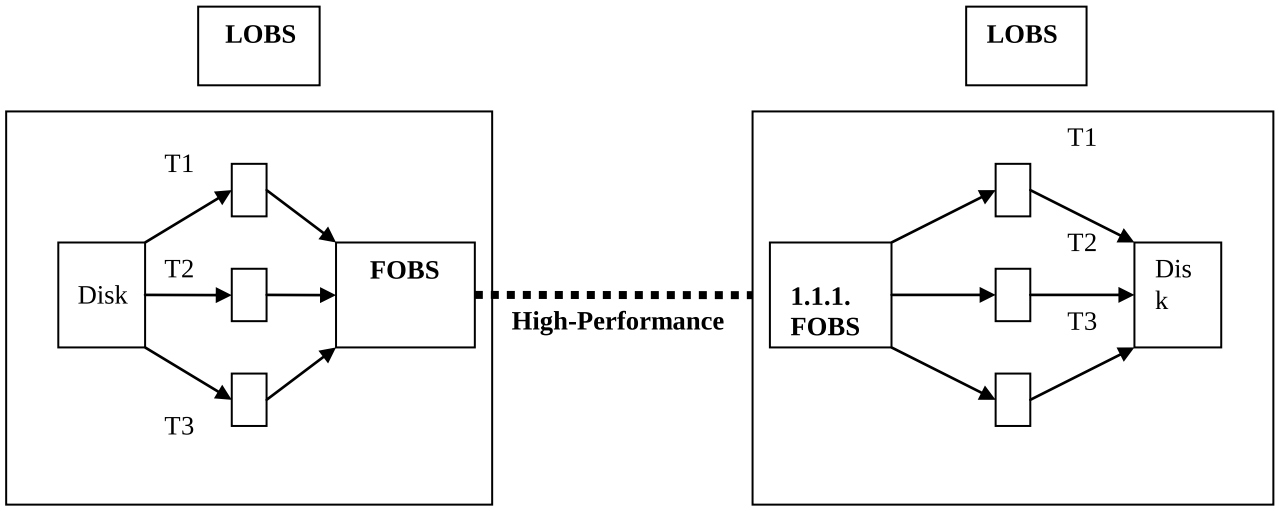

Figure 1. This figure shows the basic approach to transfer files using LOBS. As can be seen, LOBS is actually a very thin front to the FOBS data transfer mechanism.

## 4.Preliminary Performance Results

We tested our approach on two endpoints connected by the Abilene backbone network. Abilene is the high-delay, high-bandwidth network associated with the Internet2 initiative. One endpoint was an SGI Origin2000 located at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign. The other endpoint was an Intel Pentium3 Windows 2000 box located at the Laboratory for Computational Science and Engineering (LCSE) at the University of Minnesota. Both endpoints were equipped with a Gigabit Ethernet connection over an OC-12 data link. We transferred an 80 Megabyte file between these **two endpoints, experimenting with different UDP packet sizes.** The results are shown in **Figure 2. The reported bandwidth measurements include the time required to read the file off of the disk** transfer the file over the network, and write the file back to disk at the receiving end. Both endpoints had very high-performance file systems making the time required to read/write the file from/to the disk a negligible component of the overall data transfer time. For this reason, it was not necessary to use threads to perform the I/O operations in the background (and by extension it was not necessary to allocate more than one data buffer).

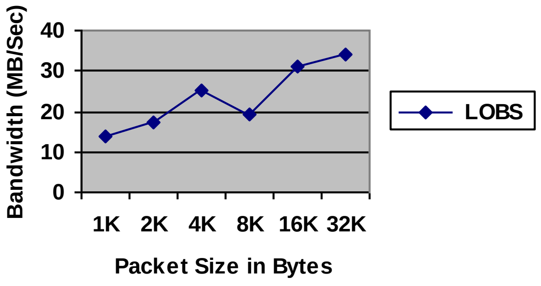

Figure 2. This figure shows the bandwidth achieved by LOBS between two sites connected by the Abilene backbone network. Performance is shown as a function of the UDP packet size.

As can be seen, LOBS was able to achieve a data transfer rate on the order of 35 Megabytes per second (280 Megabits per second). It is interesting to note that the size of the UDP data packet had a tremendous impact on the performance of the protocol. It would be interesting to measure performance using even larger buffer sizes, but 32K is the largest size supported by IRIX 6.5

We are in the process of testing the performance of LOBS over other high-performance, high-delay network connections, and the results of such testing should be available in time for the full paper.

## 5. Additional Support for Distributed-based Computing

High-performance Distributed-based applications often require higher levels of functionality than that supplied by a simple file transfer mechanism such as LOBS. To address Conclusions

In this paper, we have introduced a new file transfer mechanism designed to support high-performance applications executing in a Distributed-based environment. We have discussed the design of LOBS, and shown how it is layered on top of a lower-level data movement protocol (FOBS). We have shown that it is able to achieve data transfer rates of 35 Megabytes per second on one set of endpoints connected by the Abilene backbone network. We have also discussed our current efforts aimed at providing a significantly higher level of file transfer services for Distributed-based environments. In the full paper, we will expand the discussion of related work, will provide additional data on the performance of the mechanism, and provide a more detailed discussion of the graphical user interface and the concept of file container objects.

## References

[1] Allcock, B. Bester, Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C. Meder, S., Nefedova, V., Quesnet, D., and S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. Preprint ANL/MCS-P871-0201, Feb. 2001.

[2] Dickens, P., and B. Gropp. An Evaluation of Object-Based Data Transfers Over High Performance Networks. Submitted to the 11th High Performance Distributed Computing Conference.

[3] Dickens, P., Gropp, B., and P. Woodward. High Performance Wide Area Data Transfers Over High Performance Networks.To Appear: The 2002 International

Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems.

[4] Hobby, R. Internet2 End-to-End Performance Initiative (or Pat Pipes Are Not Enough). URL: http//www.internet2.org.

[5] Irwin, B. and M. Mathis. Web100: Facilitating High-Performance Network Use. White Paper for the Internet2 End-to-End Performance Initiative. URL:http://www.internet2.edu/e2epi/web02/p_web100.shtml

[6] Jacobson, V., Braden, R., and D. Borman. TCP Extensions for high performance. RFC 1323, May 1992.

[7] Leigh, J. et al. Adaptive Networking for Tele-Immersion. In: Proceedings of the ImmersiveProjectionTechnology/Eurographics Virtual Environments Workshop (IPT/EGVE), Stuttgart, Germany, 05/16/01-05/18/01.

[8] MacDonald and W. Barkley. Microsoft Windows 2000 TCP/IP Implementation Details. White Paper, May 2000.

[9] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018

[10] Ostermann, S., Allman, M., and H. Kruse. An Application-Level solution to TCP's Satellite Inefficiencies. Workshop on Satellite-based Information Services (WOSBIS), November, 1996.

[11] Sivakumar, H., Bailey, S., and R. Grossman. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In Proceedings of Super Computing 2000 (SC2000).

[12] Sivakumar, H., Mazzucco, M., Zhang, Q., and R. Grossman. Simple Available Bandwidth Utilization Library for High Speed Wide Area Networks. Submitted to Journal of SumperComputing

[13] URL: http://www.psc.edu/networking/perf_tune.html#intro Enabling High Performance Data Transfers on Hosts: (Notes for Users and System Administrators)

[14] URL: http://dast.nlanr.net/Articles/GettingStarted/TCP_window_size.html

[15] URL: http://dast.nlanr.net/Projects/Autobuf_v1.0/autotcp.html. Automatic TCP Window Tuning and Applications

[16] URL: http://www.globus.org

[17] URL: http://www.psc.edu/networking/all_sack.html. List of sack implementations

[18] URL: http://www.internet2.org

[19] URL: http://www.iwire.org/